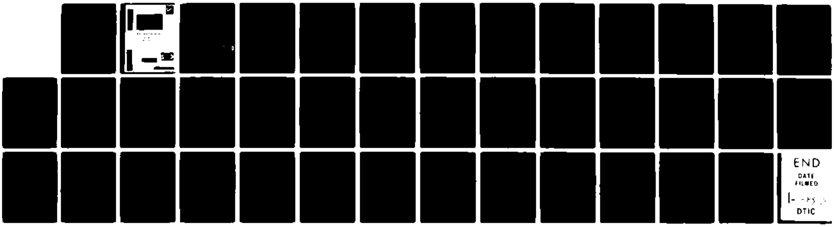
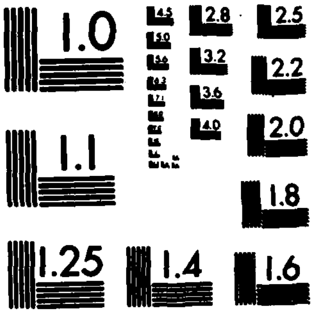
AD-A122 557 REAL TIME RESOURCE ALLOCATION IN A DISTRIBUTED SYSTEM 1/1
(U) HARVARD UNIV CAMBRIDGE MA CENTER FOR RESEARCH IN
COMPUTER TECHNOLOGY J REIF ET AL. JUN 82 TR-18-82
UNCLASSIFIED N00014-80-C-0647 F/G 5/1 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
1.1
1.25
1.4
1.6

Harvard University

Center for Research

REAL TIME RESOURCE ALLOCATION IN A DISTRIBUTED SYSTEM

John Reif

Paul Spirakis

TR-18-82

Presented at ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, August 1982

June 1982

Original version appeared as TR-06-82 in February 1982.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO. AD-A122 559

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Real Time Resource Allocation in a Distributed System

5. TYPE OF REPORT & PERIOD COVERED

Technical Report

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

John Reif
Paul Spirakis

8. CONTRACT OR GRANT NUMBER(s)

N00014-80-C-0674

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Harvard University
Cambridge, MA 02138

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Office of Navel Research
800 North Quincy Street
Arlington, VA 22217

12. REPORT DATE

June 1982

13. NUMBER OF PAGES

33

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

same as above

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

umlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

resource allocation, hypergraph matchings, real-time, synchronization, dining philosophers, scheduling, two-phase locking, reandomized algorithm.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see reverse side.

In this paper we consider a resource allocation problem which is local in the sense that the maximum number of users competing for a particular resource at any time instant is bounded and also at any time instant the maximum number of resources that a user is willing to get is bounded. The problem may be viewed as that of achieving *matchings in dynamically changing hypergraphs*, via a distributed algorithm. We show that this problem is related to the fundamental problem of *handshake communication* (which can be viewed as achieving *matchings in a dynamically changing graph*, via distributed algorithms) in that an efficient solution to each of them implies an efficient solution to the other. We provide *real-time* solutions to the resource allocation problem (that is, we give distributed algorithms with real time response). We make essential use of probabilistic techniques as first used by [Rabin, 80b], where processes are allowed to make independent probabilistic choices. On the other hand, no probability assumptions about the system behavior are made. One of our solutions assumes the existence of an underlying real-time handshake communication system, as described in [Reif, Spirakis, 81]. Our other solution is based on efficient synchronization by flag variables, which are written only by one process and read by at most one other process. The special case of equi-speed processes is first examined. Then we generalize to asynchronous processes. Applications are made to dining philosophers, scheduling and two-phase locking in databases.

REAL TIME RESOURCE ALLOCATION IN A DISTRIBUTED SYSTEM*

by

John Reif and Paul Spirakis

Harvard University

June 1982

*This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024, and the Office of Naval Research Contract N00014-80-C-0647.

ABSTRACT

In this paper we consider a resource allocation problem which is local in the sense that the maximum number of users competing for a particular resource at any time instant is bounded and also at any time instant the maximum number of resources that a user is willing to get is bounded. The problem may be viewed as that of achieving *matchings in dynamically changing hypergraphs*, via a distributed algorithm. We show that this problem is related to the fundamental problem of *handshake communication* (which can be viewed as achieving *matchings in a dynamically changing graph*, via distributed algorithms) in that an efficient solution to each of them implies an efficient solution to the other. We provide *real-time* solutions to the resource allocation problem (that is, we give distributed algorithms with real time response). We make essential use of probabilistic techniques as first used by [Rabin, 80b], where processes are allowed to make independent probabilistic choices. On the other hand, no probability assumptions about the system behavior are made. One of our solutions assumes the existence of an underlying real-time handshake communication system, as described in [Reif, Spirakis, 81]. Our other solution is based on efficient synchronization by flag variables, which are written only by one process and read by at most one other process. The special case of equi-speed processes is first examined. Then we generalize to asynchronous processes. Applications are made to dining philosophers, scheduling and two-phase locking in databases.

I. INTRODUCTION

1.1 The Resource Granting System

In this paper we consider a resource allocation problem which is *local* in the sense described in [Lynch, 1980]. The set of resources $\boldsymbol{\rho}$ and the set of user processes U may be infinite sets. However, there is a limit to the number of user processes requesting a particular resource. We assume that names of processes are integers. Resources are controlled by a set of *granting processes* R. Each granting process $j \in R$ controls a resource $\rho(j) \in \boldsymbol{\rho}$. We assume user processes communicate only with those granting processes for which they request resources. (It is easy to superimpose each granting process into a requesting user process so that $U = R$.)

A system described as above, is called a *Resource Granting System* (RGS). The goal of a RGS is to satisfy dynamically changing user requests for resource allocation. This is done in a distributed way, by only a local communication between granting and requesting processes. An *implementation* of the RGS determines the synchronization algorithms that the processes run. It is *symmetric* if these algorithms do not depend on the location of the processes in the network. At each time $t \geq 0$ the requests by user processes U are specified by an adverse *oracle* $\mathcal{A}$, (which may be an "enemy" of the resource allocation algorithm, by setting actions in the worst way to increase the respone time.) (Note that in practice no such oracle $\mathcal{A}$ may exist but instead each user process is executing some program which requires from time to time some resources to be allocated to that user process. The oracle $\mathcal{A}$ is used as an artificial device for specifying worst case situations for the resource allocation algorithm.) $\mathcal{A}$ also has the capability to select at time $t = 0$ the schedule of the speeds of all processes at all times $t \geq 0$. The oracle $\mathcal{A}$

is restricted, to allow users to keep asking for their resources until they are granted. We assume that there is a global time t totally ordering events, but processes do not have access to it.

An RGS *with priorities* is defined to be a resource granting system in which the requesting processes communicate to each resource granting process a rational number on the interval $[0,1]$, indicating the priority of the request. A priority greater than 0 indicates a request for that resource. These priorities can change dynamically. The processes of R use the priorities, that the processes of U communicate to them, to grant their controlled resource with preference to users of higher priority. This must be done in a way avoiding user starvation.

1.2 Complexity of an RGS

A process *step* consists of either an assignment of a variable, a test, a logical or arithmetic operator or a no-op. A step is considered to be a finite time interval Δ in which a single instruction is executed instantaneously at the last moment of Δ. Note that, by these semantics, there can never be read-write conflicts in the use of flag variables.

Let a process be *tame* during a time interval Δ, if for any interval $\Delta' \in [0,\infty)$, which intersects Δ and is a single step of the process, then $|\Delta'| \in [r_{min}, r_{max}]$ where r_{min}, r_{max} are fixed real constants and $0 < r_{min} \leq r_{max}$. Without loss of generality we assume that r_{max}/r_{min} is an integer. We do not assume processes always tame, however, they are supposed to be tame in the complexity analysis of response time. We require that, at no time, any granting process $i \in R$ simultaneously grant the resource $R(i)$ to more than one requesting process. We also require that, as soon as a process $j \in U$ has got all its required resources, then it can keep them only for a time interval upper bounded by a fixed parameter (resulting in a bounded number of its steps μ, if the process is tame). Let resources(i) be the set of all possible resources that a process $i \in U$ is going to ever request, and let $\text{resources}_t(i)$ be the set of resources i is requesting at time instant t. Let $k_{i,t}$ be $|\text{resources}_t(i)|$. For each time t, the *willingness* digraph G_t is defined such that if $i \in U$ and $j \in R$ and if i requests or has been granted resource $\rho(j)$ at t, then the edge $i \underset{t}{\rightarrow} j$ belongs to G_t. Let also $j \underset{t}{\rightarrow} i$ if the granting process j is willing to allocate (or has granted) its resource to user i. Let v_t be the maximum valence of the nodes $j \in R$ of G_t.

In the following we will assume at all times $t \geq 0$ that v_t and $k_{i,t}$ are above bounded by constants v,k respectively and that $v \leq k$. This does not imply anything about the cardinality of the sets resources(i) $\forall i \in U$, which could be unbounded. We also assume, as in [Lynch, 1980], that each resource allocator $j \in R$ has a set S_j available to it of size $\leq v$, containing the names of those processes willing to get the resource. We assume this to be a primitive of our system (which could be implemented by a queued message system or by other means). Finally, we restrict any oracle $\mathcal{A}$ so that as soon as it produces a request for a resource (i.e., it orders the appearance of the edge $i \underset{t}{\rightarrow} j$ in G_t for $i \in U$, $j \in R$) it has to insist on that request until i gets all its requested resources at that time. (Note that the relation $\underset{t}{\rightarrow}$ can be viewed as a time varying hypergraph with node set $\Pi = U \cup R$ and hyperedge set $E = \{\{i\} \cup \text{resource}_t(i) | \ i \in U\}$. An RGS implementation dynamically achieves matchings in this hypergraph. We allow *probabilistic* RGS implementations where processes can use independent random number generators.

Fix an RGS implementation which may be probabilistic. For each k $(0 \leq k \leq v)$ and oracle $\mathcal{A}$ let the k-*grant response* be the random variable $\gamma_{k,\mathcal{A}}$ giving the length of the minimum interval Δ required for any process $i \in U$ to have k resource requests simultaneously granted, given that i requested these resources during the entire interval Δ, with priority 1, and assuming that i and all the allocators of the resources requested by i within Δ are tame on Δ. Let the *mean* k-*grant response* be $\bar{\gamma}_k = \max\{\text{mean}\{\gamma_{k,\mathcal{A}}\}$ over all oracles $\mathcal{A}\}$. For each ε in $[0,1]$ let the ε-*error* k-*grant response* be the minimum $\gamma_k(\varepsilon)$ such that for every oracle

$$\text{Prob}\{\gamma_{k,\mathcal{A}} \leq \gamma_k(\varepsilon)\} \geq 1-\varepsilon \quad .$$

The RGS implementation is *real time* if for every $k \in \{1,\ldots,v\}$ and for every $\varepsilon \in (0,1]$, $\gamma_k(\varepsilon) > 0$ and independent of any global measure of the network (except v). The network here has as nodes the elements of Π and each edge $\{u,r\}$ denotes that $\rho(r) \in \text{resources}(u)$. Note that if the RGS implementation is real time, then $\bar{\gamma}_k$ is constant, independent of any global measure of the graph H of the network (except perhaps v).

1.3 Previous Work

[Lynch, 1980] considered the problem of allocation of resources in a distributed system. Her RGS implementation was a deterministic, non-symmetric one (processes were allowed to know the color of each resource in a coloration of the *resource graph*, i.e. the graph whose nodes are the resources and two resources i, j have an edge between them iff $\text{resource}^{-1}(i) \cap \text{resource}^{-1}(j) \neq \emptyset$), and the communication system adopted was a message system requiring buffered communication. [Lynch, 1980] achieved k-grant response of the order $\chi(H)\cdot v^{\chi(H)}\cdot\tau$ where $\chi(H)$ is the chromatic number of the resource graph and τ is the time required for process communication. Note that since resources(i) for $i \in U$ may be unbounded, in the worst case $\chi(H)$ can be as large as the number of processes $|\Pi|$.

1.4 Results of this Paper

We shall present in Section 3 a *probabilistic* implementation of an RGS which has mean k-grant response $\bar{\gamma}_k = O(kv^k \cdot \bar{\tau} \cdot \log v)$ and ε-error k-grant response

$$\gamma_k(\varepsilon) = O\left(kv^k \log\left(\frac{1}{\varepsilon}\right) \tau\left(\frac{\varepsilon}{2v}\right)\right)$$

where $\bar{\tau}$ and $\tau(\varepsilon)$ are the mean time and ε-error response required for *handshake communication* between processes (see Appendix for definitions). In [Reif, Spirakis 1981], [Reif, Spirakis 1982] handshake communication implementations were given with $\bar{\tau} = O(v^2)$ and $\tau(\varepsilon) = O(v^2 \log(1/\varepsilon))$. Note that these implementations achieve real time, thus the resulting RGS implementation is also real time with mean

$$\bar{\gamma}_k = O(kv^{k+2} \log v) \quad \text{and} \quad \gamma_k(\varepsilon) = O\left(kv^{k+2} \log\left(\frac{v}{\varepsilon}\right) \log\left(\frac{1}{\varepsilon}\right)\right) .$$

However, any other handshake communication implementation would also do.

In Section 4 we present a basic way to implement a real time RGS in both cases of equispeed and asynchronous tame processes. The implementation is probabilistic. No underlying handshake communication system is assumed. Instead, the means of synchronization between processes in U and R are *flag* variables (which are written by just one process and allowed to be read by at most one other process). The response time has mean $\bar{\gamma}_k = O(kv^{k+1})$ and $\gamma_k(\varepsilon) = O(kv^{k+1} \log(1/\varepsilon))$.

1.5 Organization of Paper

This paper is organized as follows: Section 2 contains applications of RGS to dining philosophers. scheduling, two-phase locking in databases, and real-time handshake communications. Section 3 discusses a real time RGS assuming an underlying real time handshake communication system. Section 4 discusses a real time RGS implementation by use of flags. First an implementation is discussed in which processes have the same speeds but their actions may be relatively shifted in time. At the end of Section 4 we generalize our algorithms to the case where processes are asynchronous but tame. Section 5 presents a probabilistic analysis of our algorithms. The Appendix defines handshake communication systems and their performance measures.

2. APPLICATIONS

2.1 Hasty Dining Philosophers

As a simple example, we consider an interesting RGS system which we call *"hasty dining philosophers"*. Let the requesting processes U be distinct integers, $r_0,\ldots,r_{n-1}$, and the granting processes R be $0,\ldots,n-1$ so that $U \cap R = \emptyset$. The resources are *forks* $\{\rho(0),\ldots,\rho(n-1)\}$. Each philosopher $r_j \in U$ has resources(r_j) consisting of the forks $\{\rho(r_j),\rho(r_{(j+1)\bmod n})\}$. Thus, the resource graph H is a cycle of length n. The "hasty dining philosophers" has a high level real time RGS implementation with mean 2-grant response $\bar{\gamma}_2 = O(1)$ and ε-error 2-grant response $\gamma_2(\varepsilon) = O(\log(1/\varepsilon)^2)$. The low level RGS implementation gives $\bar{\gamma}_2 = O(1)$ and $\gamma_2(\varepsilon) = O(\log(1/\varepsilon))$. Intuitively, the RGS implementation requires each philosopher r_j to be at any time granted both forks of resource(r_j) in expected constant time, but r_j must be "hasty" and relinquish these resources within constant time interval. Note that for each $i \in \{0,\ldots,n-1\}$ the granting process i can be placed within process r_i, thus resulting in essentially only n processes.

2.2 Scheduling

Suppose an acyclic digraph D is given with node out-degree $\leq k$, and in-degree $\leq v$. This graph can be separated into levels. We assume processes residing in the nodes of D can operate only after getting all their *resources* (which reside at the nodes which have no successors). Each process operates just once and in a constant time interval becomes a resource granting process, to serve the next higher level. All its successors are deleted. So, each node is initially a requesting process and after it gets all its resources once, it becomes a resource granting process. By assuming a real time RGS implementation the above system can be processed in time of the order of the depth of the digraph.

2.3 Two-Phase Locking in Databases

Two-phase locking is a concurrency control method in databases (see the survey paper of [Bernstein, Goodman 1980]) with the feature that as soon as a transaction releases a lock, it never obtains additional locks. The technique of two-phase locking produces serializable transaction executions. We propose in [Reif, Spirakis 1982B] a technique to implement two-phase locking in a distributed database, which is different from any known algorithms. In fact, it can be viewed as intermediate between the techniques of static and dynamic locking. Our underlying assumption is that transactions are allowed to act on the data only if they got all the locks requested. Let the processes of the user set U be called *transaction modules* and the processes of the set R be called *data modules*. If each transaction requests to lock at most k data modules at a time and if at most v transactions can compete for a lock at a time instant t, then a real time RGS will result in real time lock allocation per transaction. The transaction modules go through a "round" (of constant number of steps in duration) during which they communicate with all data modules they want to lock. They keep the locks they get only for constant number of steps hoping in the meanwhile to get all of them. If the round finishes and they did not succeed in getting all the locks, then they release whatever they got and try again in the next round. Given the previously stated response time $\gamma_k(\varepsilon)$ for a real-time RGS, one can now decompose the distributed system into a system of parallel servers with almost independent geometrically distributed service time, (one per transaction module). In [Reif, Spirakis, 1982B] we then analyze the transaction waiting time, throughput etc., by assuming a probability distribution in the transaction arrival rate.

2.4 Handshake Communication in Real Time Via a Real Time RGS

We can implement here handshake communication in the sense of the Appendix, assuming the existence of a real-time RGS. Assume that each of the processes $j \in R$ control just one resource called the *channel* and when they allocate this resource, we say they *open* the channel. The processes $i \in U$ are assumed to open their channel when they are granted their resource and to close their channel when the resource is removed.

Each of the processes $i \in U$ have $|\text{resource}_t(i)| \leq 1$, so as soon as any process $i \in U$ is granted one resource, process i does not compete for any other, until i releases that resource.

Given bounds v on the processes $i \in U$ competing for the same $\rho(j)$, $j \in R$ and a bound $k \leq v$ on the number of resources a user is *willing* to be granted at any time t, a real-time RGS will imply a *real-time handshake communication scheduler*, with the same time performance (see the Appendix). It is interesting to note (as we show in Section 3) that one can implement also a real time RGS by a real time distributed communication subsystem.

3. HIGH LEVEL RGS IMPLEMENTATION ASSUMING A REAL-TIME HANDSHAKE COMMUNICATION SYSTEM

3.1 The Algorithm

Our implementation of a probabilistic real time RGS is as follows: The granting processes i are always willing to communicate only to the requesting processes in the set S_i which have rights to resources $\rho(i)$ (as defined in the Introduction). The requesting processes are willing to communicate only to those granting processes whose resources they want (or have been allocated). By communication here we mean a handshake communication. We shall assume here the existence of a DCS with ε-error response $T(\varepsilon)$, as in the Appendix.

We may view the actions of the requesting processes time-sliced in *rounds*, where each round is a minimal time interval in which i communicates at least once with all k of the resource controlling processes of the resources which i wishes to obtain.

The granting processes do forever the following *grant algorithm* which is a loop, a single execution of which is called a *grant phase*:

Grant Algorithm

for process $i \in R$

```
Do forever
    begin
    [1]. Do a handshake communication with anyone of the requesting
         processes in $S_i$ and get their priorities (in $\delta$ steps).
    [2]. Probabilistically select $j \in S_i$, in $\delta$ steps. (The values of
         the priorities determine the probability of each requesting
         process to be selected as determined in Section 3.2.)
    [3]. On first handshake with the selected process $j$, say "yes" to
         $j$ allocating your resource to $j$. ($\delta$ steps)
    [4]. For $2\delta$ steps, the granting process says "no" to any requesting
         process but $j$ in any handshake and says "yes" to $j$ on any
         communication.
    [5]. On handshake with any other process than $j$, the granting process
         $i$ says "no". On first handshake communication with $j$, the
         granting process $i$ says "no" to $j$, indicating that resource
         $\rho(i)$ has been withdrawn from $j$ and ending the grant phase.
         ($\delta$ steps).
    [6]. Wait for $\lfloor w \rfloor$ steps where $w$ is randomly chosen from $[0, 6\delta']$.
    end
```

In the above algorithm we fix parameters $\delta = \lceil \tau(\varepsilon/2v)/r_{min} \rceil$ and $\delta' = \lceil \delta(r_{max}/r_{min}) \rceil$. Hence δ steps of any tame process contain at least time $\tau(\varepsilon/2v)$ and the maximum time duration of δ steps is equal to the minimum time duration of δ' steps. Thus the maximum possible length of the random wait is, at least the time length of the rest of the stages of the grant phase. Note that stages [1], [3] and [5] may take more than δ steps (with very low likelihood). This is taken into account in the analysis of performance of the algorithm.

3.2 Probabilistic Selection

We give here a very simple implementation of probabilistic selection of one out of $\leq v$ processes (using their priorities) in $O(v)$ steps as required in phase 2 of the algorithm in Section 3.1. This implementation can easily be improved to $O(\log v)$ steps (see [Reif, Spirakis 1982]).

Suppose that each resource allocator i has just a random number generator drawing uniform numbers between 0 and 1. Let $\pi_1, \pi_2, \ldots, \pi_{v'}$ (for $v' \leq v$) be the processes requesting the resource of i at the current time and let $P_{i\pi_1}, P_{i\pi_2}, \ldots, P_{i\pi_{v'}}$ be their priorities. Let $h(x) = \Sigma_{j=1}^{x} P_{i\pi_j}$, for any x in $\{0,1,2,\ldots,v'\}$. To implement the selection process of stage [2] we do the following:

[2.1] draw a random number n in $[0,1]$.

[2.2] find the process name π_x for which

$$\frac{h(x-1)}{h(v')} < n \leq \frac{h(x)}{h(v')} \quad .$$

Note that stage [2.1] takes one step and stage [2.2] takes $O(v)$ steps since we have to evaluate and compare 2 partial sums each step. Since $h(x) = h(x-1) + P_{i\pi_x}$ for any x in $\{1,\ldots,v'\}$, the current partial sum can be evaluated from the previous partial sum by a single addition.

4. REAL TIME IMPLEMENTATION BY USE OF FLAG VARIABLES

4.1 The Algorithm for the Case of Equi-Speed Processes

For simplicity, we shall temporarily assume here a fixed time instant $t_0 \geq 0$ such that for time $t \geq t_0$ all processes are executing at the same speed. (Section 4.2 drops the assumption of equi-speed processes.) However, for times $t < t_0$ in the past, the processes may be asynchronous and so, at $t \geqslant t_0$ the execution of their programs in time, though proceeding with equal speed for all processes, may be shifted (in an adverse way) relative to each other.

The communication between granting and requesting processes is done here by flag variables. To read one flag requires one of the process steps. In case of priorities, the flags are allowed to have rational values between 0 and 1, initially 0. The flags P_{ij} indicate the priority of user j with respect to resource i. In the simple case of equal priorities all flags need only be boolean. Each granting process i has for each requesting process j a special flag F_{ij} whose value indicates if the resource $\rho(i)$ is allocated to j. If j reads F_{ij} and finds it 0, then it understands that it lost the resource. The granting processes execute forever the following loop, called a *grant phase*:

Grant Algorithm

of Granting Process $i \in R$

```
Do forever
    begin
    [1].  Read the priority flags of the requesting processes in the set $S_i$.
    [2].  Probabilistically select each of the requesting processes $j \in S_i$
          according to their priorities (see Section 3.2).
    [3].  Set the flag $F_{ij}$ to 1 indicating that resource $\rho(i)$ has been
          allocated to process $j$.
    [4]   Wait for cv (non-operative) steps.
    [5]   Set the warning flag $L_{ij}$ to 1 to indicate to $j$ that he will
          loose the resource after at most 2cv steps. Wait 2cv steps.
    [6]   Set $F_{ij}$ to 0 (indicating that $i$ removes the resource).
          Erase the warning by setting $L_{ij}$ to 0.
```

```
[7]  Wait for $w$ steps where $w$ is a random integer selected
     uniformly from $[0,5cv]$.

end
```

Stages [1], [2] are required to take exactly cv steps each, where c is a small constant whose exact value can be determined by counting the maximum number of steps of the granting process per requesting process in the stages [1], [2], [3].

The grant phase takes a random number of steps, uniform in the set $\{5cv, 5cv+1, \ldots, 10cv\}$.

Each user process $j \in U$ executes continuously the following loop, a single execution of which is called a *round*.

```
Do forever
    begin
    [1].  Set $p_{ij}$ to 1 for each resource $\rho(i)$ requested by user $j$.
          ($cv$ steps)
    [2].  Poll for $cv$ steps to see which resources have been awarded to
          user $j$.  User $j$ considers the resource $\rho(i)$ awarded only
          if $\rho(i)$ has been both allocated $(F_{ij} = 1)$ and not yet warned
          $(L_{ij} = 0)$.
    [3].  If all resources requested by $j$ are awarded use them for
          $\mu$ steps.
    end
```

(See also Figures 1 and 2.)

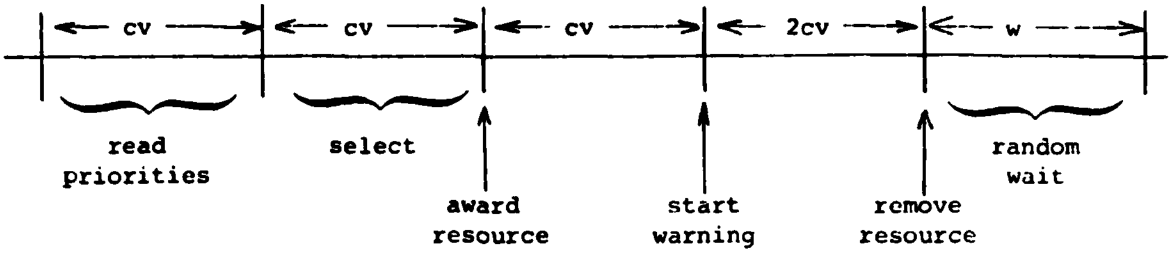


Figure 1: A *phase* of a grant process.

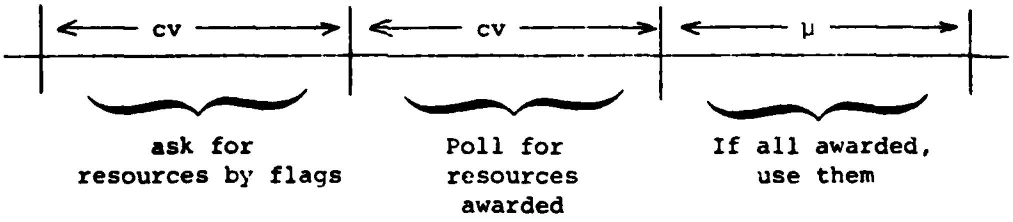


Figure 2: A *round* of a requesting process.

4.2 The Flags Implementation of the RGS for the Case of Asynchronous, Tame Processes

Grant Algorithm

for $i \in R$, for asynchronous case

It is the same as in the case of equi-speed processes. However, the stage lengths are modified as follows:

Stage [1]: takes $c'v$ steps

Stage [2]: takes $c'v$ steps

Stage [3]: 1 step as in equi-speed case

Stage [4]: takes $c'v$ steps

Stage [5]: takes $2c'v$ steps

Stage [7]: choose a random integer $w \in [0,5c''v]$

where

$$c' = \frac{r_{max}}{r_{min}} \cdot c \quad \text{and} \quad c'' = \left(\frac{r_{max}}{r_{min}}\right)^2 \cdot c \quad .$$

The *round* of $j \in U$ in asynchronous case:

Same actions as in equispeed case. The stage lengths in steps are cv, cv and

$$\mu < (cv-1)\,\frac{r_{min}}{r_{max}} \quad ,$$

for stages [1], [2], [3], respectively.

5. ANALYSIS OF THE REAL TIME RGS IMPLEMENTATION BY USE OF FLAG VARIABLES

5.1 The Case of Equispeed Processes

Note that in the equispeed case assumed here, the time is a constant multiple of the process steps. The power of the adversary is thus restricted to only a possibly malicious initial relative shift of the program counters of the various processes.

LEMMA 1.1. *For* $\mu < cv - 1$, *it is impossible for the user* j *to conclude that it has got all resources and actually some of the resources to have been removed.*

Proof. Since the polling time of j lasts only cv steps, by the time user j concludes that the last resource is allocated to j, the first allocated resource can at most be in the middle of the warning period (and hence not removed yet). Since $\mu < cv - 1$, j has then enough steps at its disposal to use the resources, before the first allocated resource is removed. □

LEMMA 1.2. *The probability that user* j *will get a particular resource in its current round is* $\leq 1/v$ *for the worst case oracles.*

Proof. Consider the subclass of oracles $\mathscr{C}$ which put maximum contention on the system. These oracles give a worst case of the response time, since contention cannot decrease the response time. However, in this case, the probability that user j will get a particular resource in its current round is at best equal to the probability that j is going to be selected by the resource allocator, so it is not more than $1/v$. □

Recall t_0 is the time instant at which the processes became synchronous. Let t_1 be any time instant after t_0. Let $\text{resources}_{t_1}(j) = \{\rho_1, \ldots, \rho_{k'}\}$, where $k' \leq k$, for a particular requesting process $j \in R$ and let $i_1, \ldots, i_{k'}$ be the resource allocators associated with those resources. In the following we consider a time interval I starting at t_1 during which the set $\text{resources}_t(j)$ for $t \in I$ is equal to $\text{resources}_{t_1}(j)$. Let Γ_{t_1} be a complete description of the system's history up to time t_1 (including the probabilistic choices of the processes up to that time). Let t_m (where $1 \leq m \leq k'$) be the first time after t_1 at which the allocator i_m starts a random wait stage. Let t_M be the maximum of all t_m's for $m = 1, \ldots, k'$ and let t_S be the maximum of the time instances at the ends of the first grant phase of processes $i_1, \ldots, i_{k'}$ after time t_M. The time interval $(t_1, t_S]$ is called a *session* Σ of processes $\{j, i_1, i_2, \ldots, i_{k'}\}$.

Let $g_j(\Sigma)$ be the probability that the process j will get all its resources in the session Σ, given any history Γ_{t_1}.

Note that a session is at most two grant phases of any resource allocator, because processes are equispeed. Note also that after t_M the effect of the history Γ_{t_1} is completely counteracted by the random waits done by the processes.

DEFINITION. Let $g_j(t_M, t_S)$ be the probability that process j will get all its resources during the interval (t_M, t_S), given any history Γ_{t_1}. Obviously $g_j(t_M, t_S) \leq g_j(\Sigma)$. Note that there is at least one complete round R of process j during a session Σ, such that R starts after t_M. Let E_m be the event "the flag $P_{i_m, j}$ of user j is seen by the resource allocator i_m in the round R and during (t_M, t_S) and j is selected by i_m." Then $g_j(t_M, t_S) \geq \text{Prob}(\bigcap_{m=1}^{k'} E_m)$.

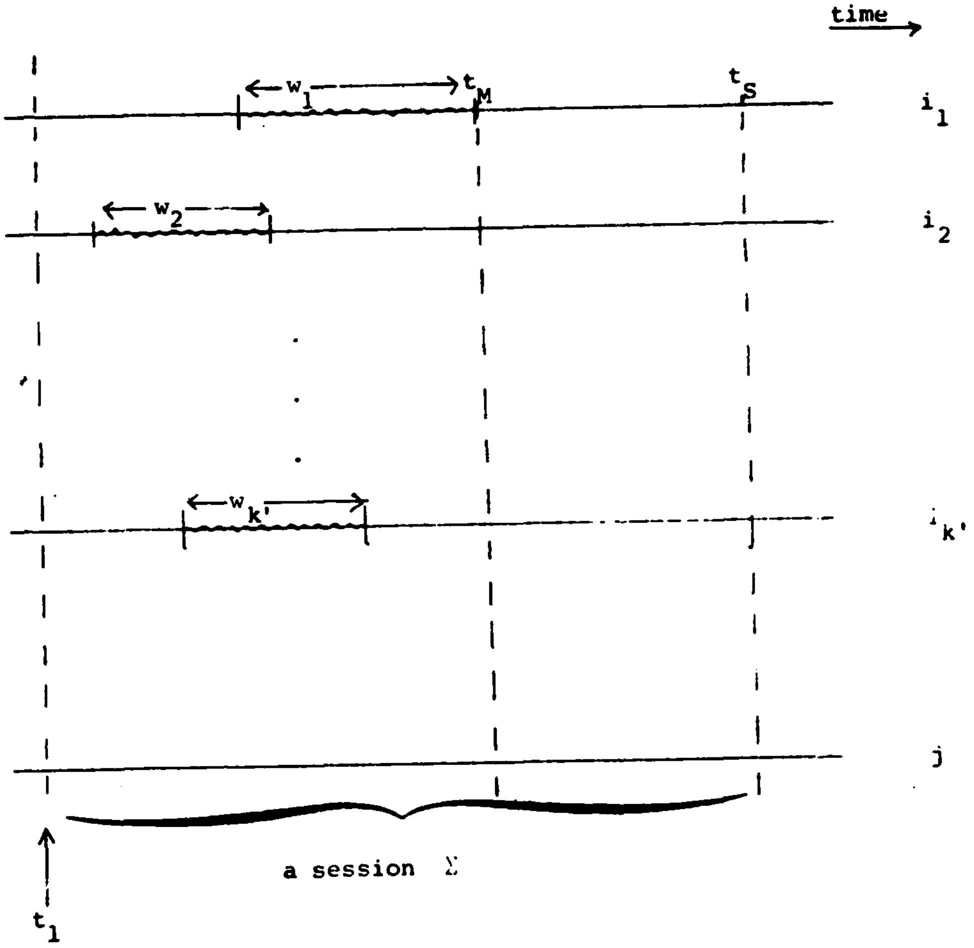


Figure 3: A *session* Σ.

LEMMA 1.3. *The events* $E_1,\dots,E_{k'}$ *are independent.*

Proof. Fix a time $t \in (t_M,t_S)$. By time t, all allocators have executed at least a random independent wait stage. The length of the wait stage is uniformly randomly chosen to take any integer value from 0 up to the length 5cv of all the other stages of a grant phase. Thus, at time t any allocator m is at each step of the non-wait part of its grant phase with equal probability. □

COROLLARY 1.3.

$$g_i(t_M,t_S) \geqslant \mathrm{Prob}(E_1)\cdot\ldots\cdot\mathrm{Prob}(E_{k'}) \quad .$$

Proof. By the independence of events E_m, $m=1,\dots,k'$ as proven in Lemma 1.3. □

LEMMA 1.4. *For any* $m \in \{1,\dots,k'\}$

$$\mathrm{prob}(E_m) \geqslant \frac{1}{10v} \quad .$$

Proof. Let E'_m be the event "the flag $P_{i_m,j}$ of user j is seen by i_m in the round R and during (t_M,t_S)." Then $\mathrm{Prob}(E_m \text{ given } E'_m) \geqslant 1/v$ since i_m selects one out of at most v processes with equiprobability and j is one of them. Also, $\mathrm{prob}(E'_m) = \mathrm{prob}$(the start of R fells into the first stage of a grant phase of i_m during (t_M,t_S)) = ratio of length of first stage divided by total length of the grant phase of i_m (by Lemma 1.3). But the total length of any grant phase is at least 5cv and at most 10cv and the length of first stage is exactly c hence

$$\mathrm{prob}(E'_m) \geqslant \frac{cv}{10cv} = \frac{1}{10} \quad .$$

Also,

$$\text{prob}(E_m) \geqslant \text{prob}(E'_m)\text{prob}(E_m \text{ given } E'_m)$$

$$\geqslant \frac{1}{10} \cdot \frac{1}{v} \quad . \qquad \square$$

By Corollary 1.3 then

COROLLARY 1.4.

$$g_i(t_M,t_S) \geqslant \frac{1}{(10v)^{k'}} \geqslant \frac{1}{(10v)^k} \qquad \textit{since} \quad k' \leqslant k.$$

Hence

$$g_i(\Sigma) \geqslant \frac{1}{(10v)^k} \quad .$$

Also, note that

$$g_i(\Sigma) \leqslant \left(\frac{1}{v}\right)^k$$

due to Lemma 1.2.

Let u be the number of sessions required for user i to be granted all its k resources in some round, given that i starts requesting them at time t_1 and assuming any history Γ_{t_1} and oracle $\mathcal{A}$. By the Corollary 1.4 and Baye's formula,

$$\text{Prob}(u{=}m) \leqslant \left(1 - \frac{1}{(10v)^k}\right)^{m-1} \frac{1}{(v)^k} \quad .$$

If $u(\varepsilon)$ is the least number such that $\text{prob}\{u > u(\varepsilon) \leqslant \varepsilon$ then

$$u(\varepsilon) = \frac{\log(\varepsilon/(10)^k)}{\log\left(1 - \frac{1}{(10v)^k}\right)} \quad .$$

It is easy to show:

PROPOSITION 1. *For every* $n \in N$,

$$\frac{1}{\log\left(1 - \frac{1}{n}\right)} > - n \quad .$$

By Proposition 1, then

$$u(\varepsilon) \leqslant k \cdot (10v)^{k} \log \frac{10}{\varepsilon} = O(kv^{k} \log \frac{1}{\varepsilon}) \quad .$$

Since each session takes $\leqslant 20cv$ steps, we have

$$\text{Prob}\{\gamma_{k,\mathcal{A}} \leqslant 20cv\ u(\varepsilon)\} \geqslant 1 - \varepsilon$$

implying ε-error response

$$\gamma_{k}(\varepsilon) = O(kv^{k+1} \log(\frac{1}{\varepsilon}))$$

and mean

$$\bar{\gamma}_{k} = O(kv^{k+1}) \quad .$$

COROLLARY 1.5. In the case of equispeed processes, our flag implementation of RGS has real time response, with ε-error response $\gamma_{k}(\varepsilon) = O(kv^{k+1} \log(\frac{1}{\varepsilon}))$ and mean $\bar{\gamma}_{k} = O(kv^{k+1})$.

5.2 The Case of Asynchronous Tame Processes

Our analysis for the case of asynchronous tame processes parallels that of the equispeed case.

LEMMA 2.1. *For* $\mu < (cv-1)(r_{min}/r_{max})$, *it is impossible for the user* $j \in U$ *to conclude that it has got all resources and actually some of the resources to have been removed.*

Proof. The polling time of i can at most be $cv\ r_{max}$, since it includes just cv steps of i. Each resource allocator waits $2c'v$ steps after setting its warning flag to 1 and then it removes the resource. Since $c' = (r_{max}/r_{min}) \cdot c$ we have that $c' \cdot v$ steps of a resource allocator are at least $c'v\ r_{min}$ time which is at least $cv\ r_{max}$ time. Hence, by the time j concludes that the last resource has been allocated to him, the allocator for the first resource granted can at most be in the middle of its warning period (in terms of steps). The maximum time corresponding to μ steps is $\mu\ r_{max} = (cv-1)r_{min}$ and the minimum time which corresponds to the remaining half steps of the warning period is at least $c'v\ r_{min} = cv\ r_{max} > (cv-1)r_{min}$. So, j has enough time at his disposal to use the resources, before the first allocated resource is removed. □

As in Lemma 1.2, we have:

LEMMA 2.2. *The probability that user* j *will get a particular resource in its current round is* $\leq 1/v$ *for the worst case oracles.*

Let t_1, Γ_{t_1}, t_M, t_S, Σ, $\{\rho_1, \ldots, \rho_k,\}$, j. I, $g_i(\Sigma)$ and $g_i(t_M, t_S)$ just as defined in Section 5.1.

A crucial difference from 5.1 is that now a session is at most two grant phases for at least one allocator and not necessarily for all of them. Again, $g_i(t_M, t_S) \leq g_i(\Sigma)$.

Note also that the minimum time duration of a grant phase is $(5c'v + 5c''v)r_{min} = 5cv(r_{max} + (r^2_{max}/r_{min}))$ and the maximum time duration of a round of a requesting process is now $(2cv + \mu)r_{max} \leq (2cv + cv(r_{min}/r_{max}))r_{max} = cv(2r_{max} + r_{min})$. This implies that a grant phase of any allocator contains at least 2 rounds of any requesting process and hence there is at least one complete round R of process j during Σ and after t_M. Again,

$g_i(t_M,t_S) \geqslant \mathrm{Prob}(\bigcap_{m=1}^{k'} E_m)$ with E_m defined as in 5.1.

LEMMA 2.3. *The events* $E_1,\ldots,E_{k'}$ *are independent.*

Proof. Fix a $t \in (t_M,t_S)$. Since $t > t_M$, by time t all allocators have each executed one (or more) random independent wait stages. The number of steps of a wait stage is a random integer chosen uniformly from 0 to 5c"v, where $c'' = (r_{max}/r_{min})^2 \cdot c$. Hence the minimum time duration of a wait stage is at least $5vc''r_{min} = 5vc(r_{max}/r_{min}) \cdot r_{max} = 5vc'r_{max}$ = the maximum time duration of the rest of the stages of the grant phase. So, in any case, the random wait can completely cover the length of the rest of the stages. So at time $t > t_M$ any allocator m is at any step x_m with equal likelihood, independently of other allocators. □

As in Corollary 1.3, we have

COROLLARY 2.3.

$$g_i(t_M,t_S) \geqslant \prod_{i=1}^{k'} \mathrm{Prob}(E_i) \quad .$$

LEMMA 2.4. *For any* $m \in \{1,\ldots,k'\}$

$$\mathrm{prob}\{E_m\} \geqslant \frac{1}{5\left(1 + \frac{r_{max}}{r_{min}}\right)} \cdot \frac{r_{min}}{r_{max}} \cdot \frac{1}{v} \quad .$$

Proof. Let E'_m be the event as defined in Lemma 1.4. Again, $\mathrm{prob}(E_m/E'_m) \geqslant 1/v$ and $\mathrm{prob}(E'_m)$ = ratio of length of first stage divided by total length of the grant phase

$$\geqslant \frac{c'vr_{\min}}{(5c'v+5c''v)r_{\max}} = \frac{1}{5\left(1+\frac{r_{\max}}{r_{\min}}\right)} \cdot \frac{r_{\min}}{r_{\max}}$$

and since $\mathrm{Prob}(E_m) \geqslant \mathrm{Prob}(E'_m) \cdot \mathrm{Prob}(E_m/E'_m)$ the lemma follows. □

Hence, by Corollary 2.3.

COROLLARY 2.4.

$$\left(\frac{1}{\lambda v}\right)^k \leqslant g_i(\Sigma) \leqslant \left(\frac{1}{v}\right)^k$$

for

$$\lambda = 5\left(1+\frac{r_{\max}}{r_{\min}}\right)\frac{r_{\max}}{r_{\min}} .$$

Let u be *defined as in case* of equispeed processes.

By following same steps as in that case, we get

$$u(\varepsilon) \leqslant k(\lambda v)^k \log\frac{\lambda}{\varepsilon} = O\left(kv^k \log\left(\frac{1}{\varepsilon}\right)\right) .$$

A session is at most two grant phases of at least one allocator. Therefore the time length of a session is bounded by $(5c'v+5c''v)r_{\max}$, so we have

$$\mathrm{Prob}\{\gamma_{k,\mathcal{A}} \leqslant 5v(c'+c'')r_{\max}\cdot u(\varepsilon)\} \geqslant 1-\varepsilon .$$

COROLLARY 2.5. *Our flag implementation of RGS in the general case of tame asynchronous processes, has real time response with*

$$\gamma_k(\varepsilon) = O\left(kv^{k+1}\log(\frac{1}{\varepsilon})\right)$$

and

$$\bar{\gamma}_k = O(kv^{k+1}) .$$

5.3 Analysis of our RGS which Uses a Handshake Communication System.

By the stated properties of DCS,

PROPOSITION 3.1.

$$\text{Prob}\left\{\text{a round length is at most } \tau\left(\frac{\varepsilon}{2v}\right)\right\} \geq \left(1 - \frac{\varepsilon}{2v}\right)^{v} \quad .$$

LEMMA 3.2. *The probability that a particular requesting process* i *will get a "yes" answer in a handshake in stage 3 is* $\leq 1/v$ *for the worst case oracles.*

Proof. Consider again the class of oracles $\mathscr{C}$ putting maximum contention in the system. □

Let t_1, t_M, t_S, Γ_{t_1}, Σ, $g_i(\Sigma)$ and $g_i(t_M,t_S)$ be just as defined in Section 5.1. Again, $g_i(t_M,t_S) \leq g_i(\Sigma)$.

Let E_m be now the event "a requesting process j gets a "yes" answer in a handshake with allocator i_m in the interval (t_M,t_S) and in the first round R of j after t_M." Note that $g_i(t_M,t_S) \geq \text{Prob}(\cap_{m=1}^{k'} E_m)$.

We show again that the events E_m are statistically independent, as in the Proof of 1.3. Again we observe that the length of a random wait has a nonzero probability of completely covering the length of the rest of the stages in a grant phase. The length of a random wait is randomly chosen from $[0,6\delta']$ So its maximum duration is at least $6\delta' r_{min} = 6\,\delta(r_{max}/r_{min})r_{min} = 6\delta\, r_{max}$ = the maximum time duration of the rest of the grant phase stages. So, Lemma 1.3 holds again, hence

COROLLARY 3.3.

$$g_i(t_M,t_S) \geq \prod_{m=1}^{k'} \text{Prob}(E_m) \quad .$$

LEMMA 3.3. *For any* $m \in \{1,\ldots,k'\}$

$$\mathrm{Prob}(E_m) \geq \frac{1}{6\left(1+\frac{r_{max}}{r_{min}}\right)} \frac{r_{min}}{r_{max}} \frac{1}{v} \quad .$$

Proof. Let E'_m be the event "the handshake of processes j and i_m will take place at the first stage of the grant phase of i_m, in the first round R of j, in the interval $[t_M,t_S]$". Then $\mathrm{Prob}(E_m|E'_m) \geq 1/v$ due to probabilistic selection of requesting processes. Also $\mathrm{prob}(E'_m) = \mathrm{prob}$ (the start of R falls into the first stage of a grant phase of i_m during time interval (t_M,t_S) = ratio of length of the first stage divided by total length of the grant phase, because by t_M process i_m has executed at least one random wait (after t_1) and hence the start of R will be any time instant of a grant phase of i_m with equal probability, independent of the history Γ_{t_1}. Hence,

$$\mathrm{prob}(E'_m) \geq \frac{\text{min length of stage 1}}{\text{max length of grant phase}}$$

$$\geq \frac{\delta r_{min}}{(6\delta+6\delta')r_{max}} \geq \frac{1}{6\left(1+\frac{r_{max}}{r_{min}}\right)} \frac{r_{min}}{r_{max}} \quad .$$

So, $\mathrm{prob}(E_m) \geq \mathrm{Prob}(E_{m'}) \cdot \mathrm{prob}(E_m/E_{m'})$

$$\geq \frac{1}{6\left(1+\frac{r_{max}}{r_{min}}\right)} \cdot \frac{r_{min}}{r_{max}} \frac{1}{v} \quad . \qquad \square$$

Due to Corollary 3.3, Lemma 3.3 and Lemma 3.2 we get

COROLLARY 3.4.

$$\left(\frac{1}{\lambda' v}\right)^k \leq g_i(\Sigma) \leq \left(\frac{1}{v}\right)^k$$

with

$$\lambda' = 6\left(1 + \frac{r_{max}}{r_{min}}\right)\frac{r_{max}}{r_{min}} \quad .$$

Let u be the number of sessions after t_1 needed for process i to get all k' resources in one round, given any history Γ_{t_1}.

The previous corollary implies

$$\text{prob}(u=m) \leqslant \left(1 - \frac{1}{(\lambda' v)^k}\right)^{m-1}\left(\frac{1}{v}\right)^k \quad .$$

For $\varepsilon \in (0,1]$, let $u(\varepsilon)$ be the minimum value such that $\text{prob}(u > u(\varepsilon)) < \varepsilon$. Then

$$u(\varepsilon) = \frac{\log(\varepsilon/(\lambda' v)^k)}{\log\left(1 - \left(\frac{1}{\lambda' v}\right)^k\right)}$$

and again (as in Section 5.2)

$$u(\varepsilon) \leqslant k(\lambda' v)^k \log\left(\frac{\lambda'}{\varepsilon}\right) \quad .$$

Note that a session Σ contains at most two grant phases of at least one resource allocator. So, the time length of a session Σ is at most

$$\begin{aligned} 2(6\delta+6\delta')r_{max} &= 12\left(\tau\left(\frac{\varepsilon}{2v}\right) + \tau\left(\frac{\varepsilon}{2v}\right)\frac{r_{max}}{r_{min}}\right)r_{max} \\ &= 12\left(1 + \frac{r_{max}}{r_{min}}\right)r_{max} \cdot \tau\left(\frac{\varepsilon}{2v}\right) \quad . \end{aligned}$$

Note that at least $\tau(\frac{\varepsilon}{2v}) = \delta$ steps are in $(t_M, t_S]$. Hence the probability that at least one complete round R is contained in $(t_M, t_S]$ is at least $(1 - \frac{\varepsilon}{2v})^v$. Now

$$\text{Prob}\left\{\gamma_{k,d} \leq (\text{time duration of a session}) \quad u\left(\frac{\varepsilon}{2}\right)\right\}$$

$$= \text{Prob}\left\{\gamma_{k,d} \leq 12\left(1 + \frac{r_{\max}}{r_{\min}}\right) r_{\max}\ \tau\left(\frac{\varepsilon}{2v}\right)\cdot u\left(\frac{\varepsilon}{2}\right)\right\}$$

$\geq$ Prob {each session contains at least one complete round R in $(t_M, t_S]$ and $u \leq u(\varepsilon/2)$}

$$\geq \left(1 - \frac{\varepsilon}{2v}\right)^v \left(1 - \frac{\varepsilon}{2}\right)$$

because the "length of a round" distribution is determined by the underlying DCS implementation independently of the number of rounds needed for a process to get all its k resources. So

$$\text{Prob}\left\{\gamma_{k,d} \leq 12\left(1 + \frac{r_{\max}}{r_{\min}}\right) r_{\max}\cdot\tau\left(\frac{\varepsilon}{2v}\right)\ u\left(\frac{\varepsilon}{2}\right)\right\}$$

$$\geq \left(1 - \frac{\varepsilon}{2}\right)^2 \geq 1 - \varepsilon$$

implying

$$\gamma_k(\varepsilon) = O\left(\tau\left(\frac{\varepsilon}{2v}\right) u\left(\frac{\varepsilon}{2}\right)\right) = O\left(kv^k \log\left(\frac{1}{\varepsilon}\right) \tau\left(\frac{\varepsilon}{2v}\right)\right)$$

which has mean

$$\bar{\gamma}_k = O(kv^k \bar{\tau} \log v) \quad .$$

Using $\tau(\varepsilon) = O(v^2 \log(1/\varepsilon))$ and $\bar{\tau} = O(v^2)$ as given in [Reif, Spirakis, 1981] we get

$$\gamma_k(\varepsilon) = O\left(v^{k+2} \log\left(\frac{1}{\varepsilon}\right) \log\left(\frac{v}{\varepsilon}\right)\right)$$

and ,

$$\bar{\gamma}_k = O(kv^{k+2} \log v) \quad .$$

Note that our response for the flag implementation is slightly better than those based on a handshake communication system, since there is no uncertainty about flag communication in each round. However, when processes are not tame, the *correctness* of the implementation by flags may be violated, while the correctness of the implementation by an underlying DCS will be preserved (because of the handshake communication which accompanies allocation or deallocation of a resource) given that the correctness of the underlying DCS is not violated when processes are not tame.

REFERENCES

Andrews, G., "Synchronizing Resources," *ACM Transactions on Programming Languages and Systems*, Vol. 3, No. 4, Oct. 81, pp. 405-430.

Angluin, D., "Local and Global Properties in Networks of Processors," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency Between Synchronous and Asynchronous Systems," *13th Annual Symposium pn Theory of Computing*, April 1981.

Bernstein, A.J., "Output Guards and Nondeterminism in Communicating Sequential Processes," *ACM Trans. on Prog. Lang. and Systems*, Vol. 2, No. 2, April 1980, pp. 234-238.

Bernstein, P., and N. Goodman, "Fundamental Algorithms for Concurrency Control in Distributed Database Systems," CCA TR. Contract No. F30603-79-0191, Cambridge, MA, 1980.

Dennis, J.B. and D.P. Misunas, "Preliminary Architecture for a Basic Dataflow Processor," *Proc. of the 2nd Annual Symposium on Computer Architecture*, ACM, IEEE, 1974, pp. 126-132.

Fischer, M.J., N.A. Lynch, J.E. Burns, and A. Borodin, "Resource Allocation with Immunity to Limited Process Failure," 19th FOCS, 1979, pp. 234-254.

Francez, N. and Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 373-379,

Hoare, C.A.R., "Communicating Sequential Processes," *Com. of ACM*, Vol. 21, No. 8, Aug. 1978, pp. 666-677.

Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem," to appear in *8th ACM Symposium on Principles of Program Languages*, Jan. 1981.

Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs," Research Report #108, Dept. of Computer Science, Yale Univ., June 1977.

Lynch, N.A., "Fast Allocation of Nearby Resources in a Distributed System," *12th Annual Symposium in Theory of Computing*, Los Angeles, California, April 1980, pp. 70-81.

Rabin, M., "N-Process Synchronization by a 4 $\log_2 N$-valued Shared Variable," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 407-410.

Rabin, M., "The Choice Coordination Problem," Mem. No. UCB/ERL M80/38, Electronics Research Lab., Univ. of California, Berkeley, Aug. 1980.

Reif, J.H., and P. Spirakis, "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time," *13th Annual ACM Symposium on Theory of Computation*, Wisconsin, 1981, pp. 133-145, also rewritten as "Real-time Synchronization of Interprocess Communications," TR-23-80, Aiken Comp. Lab., Harvard University, Cambridge, MA.

Reif, J.H., and P. Spirakis, "Unbounded Speed Variability in Distributed Communications Systems," *Ninth ACM Symposium on Principles of Programming Languages*, January 25-27, 1982A, Albuquerque, New Mexico.

Reif, J.H., and P. Spirakis, "Real Time Implementation of 2-Phase Locking by Probabilistic Techniques," to appear in 1982.

Schwarz, J., "Distributed Synchronization of Communicating Sequential Processes," DAI Research Report No. 56, Univ. of Edinburg, 1980.

.PPENDIX

Distributed (Handshake) Communication Systems (DCS)

Suppose that each process has a special resource called channel which can be in one of two states *open, closed.* A *handshake* of ·.wo processes i, j in time t is a combination of processes states at time t so that both channels of i and j are open at the same time.

Successful direct communication is a handshake of at least 1 step overlap of both processes so that the handshake relation is a matching. At any instant t no process is allowed to be handshaking with more than one other process. During the one step overlap, a message can be transmitted from one process to the other. The problem is usually to synchronize processes (via a distributed scheduler) so that they can handshake at their will, given that the means of synchronization is some low level construct (a message system, buffered communication, shared variables or flags) which does not guarantee the handshake property if used in an unsophisticated way. A distributed scheduler is called *real time* if it has the property that if two processes i, j are willing to handshake mutually for at least a constant time interval, then they will actually achieve successful direct communication during that time interval with arbitrarily small probability of error.

Formally, let $\tau(\varepsilon)$ be the smallest real number such that if two processes i,j are mutually willing to handshake for at least $\tau(\varepsilon)$ time, then they will actually succeed in 1 step overlap of open channels during that time, with probability $\geqslant 1-\varepsilon$. $\tau(\varepsilon)$ is called the ε-*error* *response* of the handshake algorithm. The *mean response* $\bar{\tau}$ of a handshake algorithm is the maximum (over all adverse speed schedules of tame processes and overall adverse communication requests subject to restrictions stated in the Introduction) of the mean time needed for two processes to handshake, from the time instant they start to be mutually willing. A real time probabilistic scheduler has $\tau(\varepsilon)$ depending only on v and not on any other global measure of the communications graph. (v is a fixed upper bound on the out-valence of the dynamic communication willingness digraph at any time instant t). We also require $\tau(\varepsilon)$ to increase at most linearly with $1/\varepsilon$. Note that such a scheduler has $\bar{\tau}$ also depending only on v.

The handshake problem has been given some attention in literature [Schwarz, 79], [Francez, Rodeh 80], [Francez, 81], [Reif, Spirakis 81], and [Reif, Spirakis 82A].

For Section 3 we require a *Distributed Communication System* (DCS) as defined above with a distributed real time probabilistic scheduler. We also require the DCS to have the following property:

If a process i is willing to communicate with $k \leqslant v$ processes for at least time $\geqslant \tau(\varepsilon)$ and if they are also willing to (handshake) communicate with i during that interval, then the probability that i will be able to communicate with all of them (in some order) within $\tau(\varepsilon)$, is $\geqslant (1-\varepsilon)^v$. Such a real time DCS was implemented in [Reif, Spirakis 81] with ε-error response

$$\tau(\varepsilon) = O(v^2 \log(1/\varepsilon))$$

and mean

$$\bar{\tau} = O(v^2) \quad .$$

DATE
FILMED

-8